Radiology: Artificial Intelligence

EDITORIAL

Magician's Corner: 3. Image Wrangling

Bradley J. Erickson, MD, PhD

From the Department of Radiology, Mayo Clinic, 200 First St SW, Rochester, MN 55905. Received July 23, 2019; revision requested September 16; revision received October 2; accepted October 8. **Address correspondence to** the author (e-mail: *bje@mayo.edu*).

Conflicts of interest are listed at the end of this article.

Radiology: Artificial Intelligence 2019; 1(6):e190126 • https://doi.org/10.1148/ryai.2019190126 • Content code: IN

"Any sufficiently advanced technology is indistinguishable from magic."

Arthur C. Clarke

In the prior articles of this series, the images used had been specially processed to apply popular deep learning libraries (1,2). This article describes why one must apply these alterations and how images are represented in computers. Image wrangling is the manipulation of images to most effectively train networks for imaging tasks. Note that "wrangle" can mean to argue, but also to herd animals, and it is this second sense of coercing and organizing the image data into a form suitable for machine learning that is meant here.

As a starting point, it should be recognized that the intensity of a pixel on an image is represented as a numerical value. The range of numerical values in the popular Joint Photographic Experts Group (JPEG) or Portable Network Graphics (PNG) format is typically from 0 to 255, which is the range that 8 binary digits (bits) permits. Medical images typically have a larger intensity range (16 bits), which allows values from 0 to 65535, or from −32768 to +32767. (There is a flag that we use to decide whether only positive or "unsigned" integers versus signed integers are used). To get color images, we use three numbers: one for each color component (red, green, and blue, or RGB). It is possible to represent images with floating point or rational numbers, and that is the common representation within deep learning libraries, and those typically range from 0 to 1.0. The reason these are not used in either JPEG or Digital Imaging and Communications in Medicine (DICOM) is they take much more space to store, require more computer power to manipulate, and without any benefit.

We can store pixels in arrays: two dimensional (2D) for the typical image that we look at, but they can be represented as three dimensional (3D) if the 2D image is part of a 3D volume or a video. Most deep learning libraries today are designed for 2D images, so we must decide how we will handle each 2D image from a 3D volume or time sequence. In many cases, the handling is trivial—we just extract each 2D image and use it individually. However, in many cases, the 3D context is important for the task at hand. In those cases, we typically need to create new types of deep learning models, a task we will discuss in a subsequent article. For now, we will focus on how to handle 2D images, and most of the lessons we learn will have obvious extensions to 3D and higher dimensions.

One of the most important decisions in image wrangling is how to handle intensities. Although most deep learning libraries are built to handle color images, most medical images are grayscale. However, some modalities do have multiple types, such as pre- and postcontrast images. In the case of MRI, we can get images that are T1-weighted, T2-weighted, postcontrast, diffusion, fluid-attenuated inversion recovery, and so forth. Here, we need to wrangle (coerce and organize) the image data to make the images consistent in size and intensity so that the algorithm learns the important differences. If you understand that RGB images really just means images that have three separate information channels, you can see that we could just put each type of image into a color channel. Let's try it!

Open the Colab notebook as follows: put *http://colab.research.google.com* into the address bar of your browser (Chrome is recommended). Then open the *ipython* notebook from the RSNA github site (File > Open Notebook > Github tab) and put "RSNA" in the search bar. From the MC-ImageWrangling folder, open the notebook entitled "ImageWrangling.ipynb." Remember to check that the runtime has GPU acceleration (Runtime > Change Runtime Type). The first cell (run cell 1) loads DICOM data selected from The Cancer Genome Atlas Glioblastoma Multiforme archive (the DICOM images are in the S1-S16.zip files). We will go through all the steps from DICOM to the machine learning in the next few cells.

Cell 2 does common preparation steps: First, it converts all the images from a DICOM series to one file in the Neuroimaging Informatics Technology Initiative (NIfTI) format (*https://nifti.nimh.nih.gov/*). NIfTI is the most popular format for image processing. Second, cell 2 performs N4 bias correction (an artifact of MRI) (3), and finally performs 3D image registration using *flirt*, FMRIB's Linear Image Registration Tool, which is an FMRIB Software Library (FSL) tool (available at *https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/*) that aligns one image set with another. In our case, *flirt* uses six degrees of freedom, but more can be specified such as scaling and skewing; for more information, see *https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/*. This step will take many minutes, so if you wish to skip, the prepared files can be used by executing cell 3. You can execute either cell 2 or cell 3, or both, but you must run at least one for this to work. There is also code that is commented out that uses another FSL tool called "bet2" which is the Brain Extraction Tool. BET was designed for 3D T1-weighted images and does not perform well on thick spin-echo images, and so it has been removed. It is included in comments in case you wish to try it.

Cell 4 loads a text file that contains the subject ID, and the first and last slice where there is contrast-enhancing tumor. This file is a comma-separated values (CSV) file

created based on the mask files that indicate which pixels have tumor. We will use the popular *Pandas* library *(https://pandas.pydata.org/)* at a later point to load and process the CSV file when we divide our files into enhancing and nonenhancing slices.

As noted before, medical images are usually stored in 16 bits, but if we want to create JPEG or PNG images, we need to convert the images to have a range from 0 to 255. The conversion to 8 bits would typically be done by applying a window width and level to define the maximum and minimum values from the 16-bit range that will be mapped to 255 and 0, respectively, with a linear ramp between. But what width and level value should we use? DICOM does have a field for width and level, but these are not always present, and they are not always good values. Therefore, we will use an algorithm to find reasonable values based on the histogram of the image. For CT, the values depend on the anatomy of interest, but for other modalities such as MRI, one typically computes the histogram and sets the range as the 5th percentile value to the 95th percentile value, excluding values less than 30, the latter being done to ignore values that are air or background.

Cell 5 creates new versions of the T1-weighted precontrast ("T1"), T1-weighted postcontrast ("GAD"), and T2-weighted ("T2") images that range in intensity from 0 to 255, and where the 0 intensity value maps to the 5th percentile value of the original DICOM image and 255 maps to the 95th percentile value. Feel free to adjust these values to see the impact on performance. Cell 5 also introduces us to Python functions, defined in the two lines in this cell that start with "def." Functions are useful when you have the same set of computations applied to data many times. Rather than repeating a block of code, one creates a function and then calls it with different parameters, for example, to apply the same function to different images.

There is a very popular matrix computation library called NumPy *(https://numpy.org/)* that is used by nearly all DL packages, and which we will use. Each of the MRI series was stored as 3D NumPy arrays (also called *ndarray* because NumPy allows *n*-dimensional arrays). NumPy has many useful functions, including ones that find the histogram values and map them to 8 bits. If we want to ignore values less than 20 on the MR images, we create new arrays that exclude values less than our minimum value (we use a variable called MIN_MR in the actual code):

$$above = np.where(image > MIN\_MR)$$

"*np*" refers to the NumPy library, and the "*where*" function finds all the values where the expression in the parenthesis is met. Next we sort the values in this array:

$$sorted\_im = np.sort(above, axis = None)$$

And now that the values are sorted, we can find the pixel values at the 5th and 95th percentiles by looking 5% *and* 95% of the way into this array:

$$start = int(sorted\_im.size/20)\#\ 20 = 5\%$$

$$end = int(sorted\_im.size * 19/20)\#\ 95\%$$

Note that *start* and *end* are indexes into the array, not the pixel values that we see. To get those, we do this:

$$start\_val = sorted\_im[start]$$

$$end\_val = sorted\_im[end]$$

Finally, we need to set values below the minimum intensity to a set minimum value, and those above the maximum intensity to a set maximum value so they don't go out of bounds, subtract the minimum value from every value in the image so its lowest intensity is 0, and then divide by the maximum value and multiply by 255 so the intensity range is 0 to 255. Note that cell 4 creates the functions that we will call in the next cell, and then these functions will do their work.

In cell 6, we will loop through the subjects and the three MRI series to scale them, and then insert each of the 2D images into the red, green, or blue color channels. In this example, we will use the library *imagio*, which is able to take the NumPy arrays and create the files. We will create PNG files because JPEG images, by default, apply lossy compression. Because this may destroy textures, we want the deep learning algorithms to "see," we will use PNG, which is lossless by default.

Cell 4 will display some images just so you can see the result. Because we encoded the postcontrast images into the red channel, contrast-enhancing tissues should have a red tint. Cerebrospinal fluid will be blue because it is dark on both T1-weighted precontrast and T1-weighted postcontrast images while being bright on T2-weighted images, and we put the T2-weighted image into the blue channel. Fat should be mostly white, because it is bright on all image types.

Up to this point we have dealt with image intensity; now we need to deal with their size. In many cases, we need to alter the X, Y, and possibly Z size of images. NumPy does not have functions for this, but the Pillow library does *(https://python-pillow.org/).* Cell 5 shows how we can resize the CT images down to 256 pixels in the X and Y dimension. At a later point, we will discuss the use of pretrained networks, and the most popular of these assume the images are 224 × 224, which is why one must know how to resize images.

There are times when one must reduce the size of the image matrix to make the problem computationally tractable. For instance, CT images are usually 512 × 512, and that may be too large. One can "decimate" the 512 matrix by taking every other pixel in X and Y directions, reducing the spatial resolution in half. If the structure of interest is small enough, one could preserve resolution, and instead crop the CT to a smaller matrix. The same is true for intensity resolution—most AI algorithms only operate on 8 bits of intensity resolution so we must apply window width and level operations. If the problem is to find

a stroke at head CT, narrow window at the right level is critical, while looking for lung nodules will likely perform best with "lung windows." The best approach to these image manipulation decisions will depend on the nature of the problem and is an example of why it is critical to have physicians involved in the development of AI tools. Similarly, the decision about whether image registration should be done before or after operations such as cropping and masking will also depend on the problem. At times, one needs the full image to achieve the best alignment, while at other times, removing some structures will enable registration to best align the component of interest.

Now that we have discussed many of the practical aspects of image intensity and size manipulation, there are times when some more advanced image processing may be desirable. Many popular image processing algorithms are available in both NumPy and Pillow, including filter functions, as well as more advanced functions such as morphologic processing. Some examples using the *fslmath* function (from the FSL library) are shown in cell 6, although we will not execute them in this example. The examples are provided only to show you that they are available, and a reading of the FSL website will help you find functions you might wish to try.

Cell 7 starts to apply the image wrangling we have done. In this example, we use the information in our CSV file to identify the slices that have enhancing tumor and those that don't. We then use the *fastai* classifier code from the prior articles to train a classifier to find slices that have enhancement versus those that don't. Note that we are not actually "segmenting" the voxels that are enhancing, but just deciding which slices have enhancing tumor versus those that don't. Note the subtle challenge is that *normal* enhancing tissues (eg, dura, nasal mucosa, pituitary gland, vessels) should *not* be classified as enhancing tumor. You will see that this simple classifier does an admirable job of identifying those slices.

At this point, you should see many opportunities to try different handling of the images, such as applying intensity thresholds, applying BET2 to remove nonbrain tissue, or applying a noise-reduction filter. Such steps can be critical elements in optimizing a classifier's performance, but often they are only tersely described. As you will see through your experiments, these steps can have significant impact.

Disclosures of Conflicts of Interest: B.J.E. Activities related to the present article: disclosed no relevant relationships. Activities not related to the present article: paid board member (and founder) of FlowSIGMA. Other relationships: disclosed no relevant relationships.

References

1. Erickson BJ. Magician's corner: how to start learning machine learning. Radiol Artif Intell 2019;1(4):e190072.
2. Erickson BJ. Magician's corner: 2. Optimizing a simple image classifier. Radiol Artif Intell 2019;1(5):e190113.
3. Tustison NJ, Avants BB, Cook PA, et al. N4ITK: improved N3 bias correction. IEEE Trans Med Imaging 2010;29(6):1310–1320.